ON INTEGER LINEAR PROGRAMMING

CHARLES WENDELL HOBART

ON INTEGER LINEAR PROGRAMMING

by

Charles Wendell Hobart
Captain, United States Army
B.S. Ed., Northeastern University, 1961

Submitted in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
June 1968

ABSTRACT

A survey of the methods of solving the integer program, $\max \sum_{j=1}^{n} c_j x_j$ subject to $\sum_{j=1}^{n} a_{ij} x_j = b_i$ $(i=1,\ldots,m)$ and $x_j \geq 0$ and integer $(j=1,\ldots,n)$ is presented. Emphasis is placed on methods developed since 1960 with many as yet unpublished methods presented. Examples are given for the unpublished methods.

TABLE OF CONTENTS

1. Introduction and Brief History

This is a survey of the progress that has been made in the solving of linear programming problems when variables must take on integer values. The field is divided into pure integer programming, when all the variables must be integers, and mixed integer programming, when only specified variables must be integers. The subject is also known as discrete programming or integer linear programming abbreviated to integer programming which appears to be the preferred term.

The integer programming problem is to

$$(1) \quad \text{Maximize} \sum_{j=1}^{n} c_j x_j$$

$$\text{Subject to} \sum_{j=1}^{n} a_{ij} x_j = b_i \quad (i=1,\ldots,m)$$

$$x_j \geq 0 \text{ and integer} \quad (j=1,\ldots,n),$$

where for simplicity and without loss in generality we assume all a_{ij}, c_j, b_i are integer valued constants.

There are five approaches capable of solving problems of this type: cutting plane methods, primal methods, branch and bound methods, partial enumeration methods, and dynamic programming procedures. These various methods will be discussed in the succeeding sections.

Although an attempt is made in this thesis to delineate the various methods of solving integer linear programs, there are no clear cut divisions. Many authors have grouped branch and bound, branch and exclude, and dynamic programming techniques into a broad category of partial enumeration methods.

If a solution of (1) as a linear program does not have the required integer properties then methods for achieving it must be invoked. A basic approach common to most methods is to successively deduce supplementary linear constraints from the linear constraints of (1) and the integer requirements until a new linear program is obtained whose solution satisfies the integer requirements.

The idea of new constraint generation appears to have been first advanced by Dantzig, Fulkerson, and Johnson [7] in 1954 in their work on the traveling salesman problem. In 1958 Gomory [11] developed a systematic method for new constraint generation in his "Method of Integer Forms" for solving pure integer programming problems in which all variables are required to be integer valued. This method guarantees that an integer solution is found (if any exist) in a finite number of steps. In 1960 Gomory devised another method, the "All Integer Method", which requires only addition and subtraction in computation provided the a_{ij} and c_j in (1) are integer valued. The above algorithms are "dual methods" and as such, no feasible solution to the problem of interest is obtained until an optimal solution is found. There is, however, a method which provides primal integer solutions. This method is attributed to Gomory but literature on this is scarce. Young [21] has also presented a similar algorithm for primal integer solutions.

The field of mixed integer programming is less far advanced. Gomory [12] has extended his method of integer

forms to deal with continuous as well as integer variables. A dual decomposition approach due to Benders [4] has been used to remedy the situation of a mixed program. In this approach the problem is partitioned and every stage of the computation involves the solution of two subproblems, a pure integer problem and a linear programming problem.

Other approaches to solving pure and mixed problems have been proposed. In 1960 Land and Doig [20] developed a branch and bound technique. In 1968 Greenberg and Hegerich [19] developed a branch and exclude algorithm for the special "knapsack problem" which was extended by Greenberg [15] to the more general problem.

Partial enumeration techniques have also been devised by Balas [1] and [2] in his "Additive Algorithm" and "Discrete Programming by Filter Method", Goeffrion [8] in his "Reformulation of Balas' Algorithm for Integer Programming", and by Glover [10] in "A Multiphase Dual Algorithm for the Zero-One Integer Programming Problem". An enumerative scheme for computation of knapsack functions by Greenberg [18] is presented in this paper.

Dynamic programming procedures have also been devised to solve linear programming problems. Two methods by Greenberg [16] and [17] are presented; one for the knapsack problem and the second a more generalized problem of integer programming.

2. Cutting Plane Methods

Before discussing the particular algorithms of Gomory it is appropriate to define some terms which will be used throughout the discussion of the algorithms. A "feasible solution" is a solution where all variables that are required to be integer are integer and all variables that have to be non-negative are non-negative. Since Gomory's methods are expressed more neatly in terms of an objective function to be maximized the objective function will be

$$(2) \quad \text{Maximize } x_0 = \sum_{j=1}^{n} c_j x_j \quad .$$

An "optimal feasible solution" is a feasible solution which yields the largest value of x_0.

In the original method for pure integer programming, the "Method of Integer Forms", one starts by solving the problem by ignoring the requirement that the variables must be integer. If this produces a solution in integers, then the problem is solved. Otherwise one considers the expression for some basic fractional variable in terms of the non-basic variables. Let this be

$$x_s = a_{s0} - \sum_{j=1}^{n} a_{sj} t_j$$

where t_j denotes the j^{th} non-basic variable.

Now write the coefficients a_{sj} in the form

$$a_{sj} = n_{sj} + f_{sj}$$

where the n_{sj} are all integers (positive or negative) and the f_{sj} are fractions such that: $0 \leq f_{sj} < 1$.

Next consider the expression:

$$s = -f_{s0} + \sum_{j=1}^{n} f_{sj}t_j \quad .$$

This expression must be an integer since it differs from $-x_s$ by an integer. Also, it cannot be less than $-f_{s0}$ since the expression $\sum_{j=1}^{n} f_{sj}t_j$ is non-negative. So s is a non-negative integer and can be introduced as a new variable of our problem. Its value at the current trial solution is $-f_{s0}$. So the constraint that $s \geq 0$ "cuts off" this solution and enables us to continue optimizing using the dual simplex method.

Gomory's second method for pure integer programming, the "All Integer Method", proceeds differently. One does not start by solving the problem in continuous variables since this method involves keeping the coefficients integral. One starts by making the problem dual feasible, perhaps by adding an artificial constraint that the sum of the non-basic variables is less than or equal to some arbitrarily large number. After this every pivotal row is a new cut generated in such a way as to make the pivot equal to -1. This ensures that the integral tableau remains integral. The procedure for generating pivotal rows is as follows: Suppose x_s is some variable that is negative in the current trial solution, then express x_s in terms of the non-basic variables as

$$x_s = a_{s0} - \sum_{j=1}^{n} a_{sj}t_j$$

where a_{s0} is negative.

Let λ be any positive quantity and write

$$a_{sj} = \lambda n_{sj} + f_{sj}$$

where n_{sj} is an integer and

$$0 \leq f_{sj} < \lambda \quad .$$

Consider the expression:

$$s = n_{s0} - \sum_{j=1}^{n} n_{sj} t_j \quad .$$

This expression must be integral and positive since

$$f_{s0} - \sum_{j=1}^{n} f_{sj} t_j \leq f_{s0} < \lambda \quad .$$

For any value of λ , $n_{s0} \leq -1$ if $a_{s0} < 0$. So the new constraint that $s \geq 0$ "cuts off" the current trial solution. The pivot can be made equal to -1 by taking λ large enough since for arbitrarily large λ all n_{sj} will be 0 or -1 depending on whether a_{sj} is non-negative or not.

Just as constraints for the Method of Integer Forms can be generated from linear combinations of rows provided it is integral, constraints for the All-Integer Method can be generated from linear combinations of rows provided it is non-negative.

The third and final cutting plane method which we will consider is Gomory's method for the mixed integer problem. This method is similar in part to the method of Integer Forms in that one first solves the problem ignoring the requirement that any variables must be integral. If some variable that must be integer, say x_s, equals $n_{s0} + f_{s0}$, where n_{s0} is a non-negative integer and $0 \leq f_{s0} < 1$, then x_s can be expressed in terms of the current non-basic variables as

$$x_s = n_{s0} + f_{s0} + \sum_{+} a_{sj} t_j + \sum_{-} b_{sj} t_j$$

where the non-basic variables t_j are divided into two classes such that all $a_{sj} \geq 0$ and all $b_{sj} \geq 0$.

Consider the expression:

$$s = -f_{s0} + \sum_{+} a_{sj} t_j + \frac{f_{s0}}{1-f_{s0}} \sum_{-} b_{sj} t_j .$$

This expression is negative at the current trial solution but is must be non-negative for any integral value of x_s. This is because if $x_s - n_{s0} \leq 0$, then

$$s \geq -f_{s0} + \sum_{+} a_{sj} t_j - \sum_{-} b_{sj} t_j = -(x_s - n_{s0}) \geq 0$$

and if $x_s - n_{s0} \geq 1$, then

$$\frac{1-f_{s0}}{f_{s0}} s \geq -(1-f_{s0}) - \sum_{+} a_{sj} t_j + \sum_{-} b_{sj} t_j = x_s - n_{s0} \geq 0 \quad .$$

So $s \geq 0$ and this is a valid cut. If any of the non-basic variables must be integral then an expression for x_s plus an integer combination of the non-basic integer valued variables, such that $a_{sj} \leq f_{s0}$ or $b_{sj} \leq (1-f_{s0})$ for all such variables can be used.

3. Primal Methods

The primal approach involves progressing from one feasible solution to another with a larger value of the objective function or else proving that no better solution can exist. A method, attributed to Gomory, and conveyed to the author in private conversation with Greenberg is a method of solution to the general problem

$$(3)\quad \text{Maximize} \sum_{j=1}^{n} c_j x_j$$

$$\text{Subject to} \sum_{j=1}^{n} a_{ij} x_j = b_i \quad (i=1,\ldots,m)$$

$$x_j \geq 0 \text{ and integer.}$$

Along with the discussion of the algorithm will be an illustrative example which is

$$(4)\quad \text{Maximize } x_1 + x_2$$

$$\text{Subject to } 3x_1 + 2x_2 \leq 7$$

$$x_1, x_2 \geq 0 \text{ and integer.}$$

Adding a slack variable, S, to our constraint equation and making a substitution $x_1 = \bar{x}_1$ and $x_2 = \bar{x}_2$, (4) may be rewritten as

$$(5)\quad \text{Maximize } \bar{x}_1 + \bar{x}_2$$

$$\text{Subject to } 3\bar{x}_1 + 2\bar{x}_2 + S = 7$$

$$-\bar{x}_1 + x_1 = 0$$

$$-\bar{x}_2 + x_2 = 0$$

$$x_1, x_2 \geq 0 \text{ and integer.}$$

Set up the first tableau using (5). This is the following

$\bar{x}_1$	$\bar{x}_2$	S	x_1	x_2	b
3	2	1	0	0	7
-1	0	0	1	0	0
0	-1	0	0	1	0
1	1	0	0	0	0

with the last row of the tableau the coefficients of the objective function. Since this is a maximize problem, the pivot column will be the column where the coefficient of the objective function is the greatest. In this case our pivot column will be either $\bar{x}_1$ or $\bar{x}_2$. Arbitrarily select the $\bar{x}_1$ column. The criteria used for selection of the pivot row is

$$\min_{a_{i\sigma}>0} \left(\frac{b_i}{a_i}\right) \text{ where } \sigma \text{ is the pivot column.}$$

In our problem the pivot element is 3.

We divide the pivot row by the pivot element giving

(6) $\quad 1 \quad \frac{2}{3} \quad \frac{1}{3} \quad 0 \quad 0 \quad \frac{7}{3}$

Rewriting (6) gives us

$$\bar{x}_1 + \frac{2}{3}x_2 + \frac{1}{3}S = \frac{7}{3}$$

or

$$(7) \quad \frac{2}{3}\bar{x}_2 + \frac{1}{3}S - \frac{1}{3} = 2 - \bar{x}_1 .$$

Since the left side of (7) is ≥ 0 then the cut is

$$\frac{2}{3}\bar{x}_2 + \frac{1}{3}S \geq \frac{1}{3} .$$

Using (6), we take the greatest integer $\leq$ the elements of the pivot row giving us

$$1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 2 \ .$$

We must always keep the slack variable, S in our case, and the original variables, x_1 and x_2, in the basis.

Since the right side of (7) implies $\bar{x}_1 \leq 2$, we must add a slack variable, say t_1, giving us

(8) $\bar{x}_1 + t_1 = 2$.

We now eliminate our pivot variable $\bar{x}_1$ from the equations in our tableau by using (8).

Next we determine if this is an optimal solution. If not we must repeat this procedure. If our pivot element is 1 at any step, we must modify the procedure by substituting dummy variables for the variable we want to keep in the basis.

Continuing with our example and using (8) our next tableau is

t_1	$\bar{x}_2$	S	x_1	x_2	b
-3	2	1	0	0	1
1	0	0	1	0	2
0	-1	0	0	1	0
-1	1	0	0	0	2

.

Note that the $\bar{x}_1$ column has been eliminated.

Our next iteration allows us to eliminate $\bar{x}_2$ from the basis by rewriting line one of the tableau as

$$-\frac{3}{2}t_1 + \bar{x}_2 + \frac{1}{2}S = \frac{1}{2} \quad \text{or} \quad -2t_1 + \bar{x}_2 = 0 \ .$$

Substituting $\bar{x}_2 = 2t_1 - t_2$ in our tableau gives

t_1	t_2	s	x_1	x_2	b
1	-2	1	0	0	1
1	0	0	1	0	2
-2	1	0	0	1	0
1	-1	0	0	0	2 .

Using the column headed by t_1 as our pivot column now requires the use of our modified procedure since our pivot element is 1.

Writing line one of the above tableau, not including S, gives

$$t_1 - 2t_2 \leq 1 \quad .$$

Adding a slack variable t_3 gives

$$t_1 - 2t_2 + t_3 = 1$$

or $$t_1 = 1 + 2t_2 - t_3 \quad .$$

Substituting this in our previous tableau and eliminating the t_1 column gives

t_3	t_2	S	x_1	x_2	b
-1	0	1	0	0	0
-1	2	0	1	0	1
2	-3	0	0	1	2
-1	1	0	0	0	3

Using the column headed by t_2 as our pivot column allows us to eliminate t_2.

Writing line two of the above tableau after dividing by the pivot element gives

$$-\frac{1}{2}t_3 + t_2 + \frac{1}{2}x_1 = \frac{1}{2}$$

and using our procedure gives

$$-t_3 + t_2 \leq 0 \quad .$$

Adding a slack variable, t_4 to this equation allows us to substitute

$$t_2 = t_3 - t_4$$

in our tableau giving

t_3	t_4	S	x_1	x_2	b
-1	0	1	0	0	0
1	-2	0	1	0	1
-1	3	0	0	1	2
0	-1	0	0	0	3

which is an optimal solution with $x_1 = 1$, $x_2 = 2$, $s = 0$, and 3 as the value of the objective function.

In this problem there is an alternate solution indicated by the 0 element in the last row of the t_3 column in the previous tableau.

Continuing with our procedure and using the t_3 column as the pivot column, we obtain an alternate optimal solution of $x_1 = 0$, $x_2 = 3$, $s = 1$, and 3 as the value of the objective function.

4. Branch and Bound Methods

A third approach to the integer programming problem is the branch and bound technique. This involves setting up a tree of linear programming problems. The root of the tree is the problem in which all integrality constraints are ignored. If the solution satisfies the integer requirements then the problem is solved. Otherwise a branching procedure is employed to obtain an integer solution to the problem.

Three methods will be discussed in this section. The first is the Land and Doig procedure [20] which was published in 1960, the second method by Greenberg and Hegerich [19] which was devised in 1968 and applied to the knapsack problem, and the third method by Greenberg [15] in 1968 which was an extension of [19] to the more generalized problem.

The general approach of the Land and Doig procedure is to solve the integer program ignoring the integrality constraints. If the solution $X^0 = (y_0, x_1^0, \ldots, x_{m+n}^0)$ satisfies the integrality constraints the problem is solved. If not then the present value y_0 of the objective function provides an upper bound on the value of the optimal solution to the integer program.

A variable x_k which is required to be integer valued but currently is not, is considered. The variable must be forced to take on an integer value hence decreased to $[x_k^0]$ or increased to at least $[x_k^0] + 1$ where $[x_k^0]$ is the greatest integer less than or equal to x_k. The entire range of possible integer values for x_k may be found by solving the

two linear programs max x_k and min x_k subject to the constraints of (2). For each possible integer value in the range the max y_0 subject to the constraints of (2) with x_k fixed can also be computed by linear programming. Thus, all possibilities in the form of a directed tree can be enumerated with the root of the tree, with rooted node 0, corresponding to X^0, and directed branches (0,i) with node i corresponding to the solution X^i of the linear program max y_0 subject to the constraints of (2) and x_k set at some specific integer value, say $x_k^{(i)}$. From each new node the same procedure can be applied. Continuing until no further branching is possible, every terminal node will either correspond to a feasible integer solution or to some sequence of integer valued choices which do not satisfy (2). The feasible node with max y_0 provides the optimal solution.

The Land and Doig procedure involves this enumeration but attempts to search only a subset of the possibly immense tree described. The idea is to develop only that part of the tree which contains the optimal solution. Any node in the directed tree must have a value of y_0 no greater than its predecessors' since branching is imposing an additional variable to some integer value, i.e., imposing an additional constraint. Also if at any node X^i of the tree, x_p is integer restricted and $x_p^{(i)}$ is not integer then the integer values for x_p which will result in the least decrease in y_0 after branching must be either $[x_p^{(i)}]$ or $[x_p^{(i)}] + 1$. Thus we have reduced the total number of branches that need be explored.

Another approach to the branch and bound or more precisely branch and exclude method is that of Greenberg and Hegerich. Their original work was with the knapsack problem and later extended by Greenberg to the generalized integer programming problem.

They present a branch and bound algorithm and a branch and exclude procedure which is used to solve the problem

$$(9) \quad \text{Maximize} \quad \sum_{i=1}^{n} v_i x_i$$

$$\text{Subject to} \quad \sum_{i=1}^{n} w_i x_i \leq W$$

$$x_i = 0,1 \quad (i=1,\ldots,n) \quad .$$

It is assumed that v_i and w_i are positive integers and $v_1/w_1 \geq v_2/w_2 \geq \cdots \geq v_n/w_n$.

The problem is first solved as shown by Dantzig [6] where the $0 \leq x_i \leq 1$. This solution is given by

$$x_i = 1 \quad \text{if } i < r$$

$$x_i = 0 \quad \text{if } i > r$$

$$x_r = \frac{W - \sum_{i < r} w_i}{w_r}$$

where r is the least integer $(0 \leq r \leq n)$ for which $\sum_{i \leq r} w_i \geq W$. If no r exists then all $x_i = 1$. If $x_r = 0$, then we have the optimal solution to (9).

If x_r is fractional the value of the objective function is $Z(1) = \sum_{i < r} v_i + v_r x_r$. $Z(n)$ is the value of the objective

function at node n. The branch and bound method is

1. Label node 1 with Z(1).

2(a). Find the terminal node with the largest value of Z(n). This is the node at which the next branching will take place. Any node (except one) contains the effect of assigning values to variables and solving (9) with the assigned values of the variables added as constraints.

(b). If the solution at node n has all variables assigned an integer value, this is the optimal solution to (9). If not, proceed to step 3.

3(a). Set n = n + 1 and some unassigned variable, say x_t to 0. Solve (9) with all assigned variables added as constraints. Label node n with the value Z(n) and proceed to 3(b).

(b). Set n = n + 1 and x_t = 1. Solve (9) with all assigned variables added as constraints. Label node n with the value Z(n) and go to step 2.

The criteria used to select x_t in step 3(a) is to take the variable that is fractional at node n.

As is common with most branch and bound techniques they require large amounts of computer storage. The branch and exclude algorithm which is described next is designed to eliminate the problem of storage. The algorithm first finds

an obvious integer solution to the constraints of (9). This solution is taken as a lower bound to the optimal solution. A branch of a tree is developed and each part explored until the lower bound is reached or until a new feasible solution is found that represents a larger lower bound. Then backtrack and develop new branches of the tree developing possibly larger lower bounds. Further branching is excluded when the lower bound is reached. The algorithm stops when all new branches are excluded. The only information that is stored is the current lower bound solution and the branch routing. At the end, the lower bound solution is optimal.

Before the algorithm is presented certain terms must be defined.

Define $S(x_0,x_1,\ldots,x_n)$ as the current lower bound solution where the x_i are all given and $x_0 = \sum_{i=1}^{n} v_i x_i$.

Define $X(x_i,x_2,\ldots,x_n)$ to indicate assigned variables. A value $x_i = 2$ (or any number not equal to zero or one) indicates the variable is unassigned. An assigned variable will have the value zero or one.

Define R(L) as the index of the L^{th} assigned variable.

Define Z(L) as the value of the objective function at the L^{th} level of a branch.

The solution to (9), with L assigned components of X added as constraints is

$$x_i = 1 \text{ if } i < r,\ i \neq R(j) \quad (j=1,\ldots,L)$$

$$x_i = 0 \text{ if } i > r,\ i \neq R(j) \quad (j=1,\ldots,L)$$

(10) $x_{R(j)} = 0$ or 1 $(j=1,\ldots,L)$ depending on the assignment

$$x_r = (W - \sum_{i=1}^{L} w_{R(i)} x_{R(i)} - \sum_{i \in M(L)} w_i)/w_r$$

$$Z(L+1) = \sum_{i=1}^{L} v_{R(i)} x_{R(i)} + \sum_{i \in M(L)} v_i + v_r x_r$$

where the set M is given by

$$M(L) = \{i \mid i < r,\ i \neq R(j) \quad (j=1,\ldots,L)\}$$

and r is the least integer $(0 \leq r \leq N)$ for which

$$\sum_{i \in M(L)}^{L} w_i + w_r \geq W - \sum_{i=1}^{L} w_{R(i)} x_{R(i)} \quad .$$

If no r exists we have all $x_i = 1$ for $i \neq R(j)$ $(j=1,\ldots,L)$. A lower bound to the solution of (9) is given by

$$x(L) = \sum_{i=1}^{L} v_{R(i)} x_{R(i)} + \sum_{i \in M(L)} v_i \quad .$$

The algorithm is as follows:

1. Set L = 1 and all components of X to two.

2. Solve (9) with $0 \leq x_i \leq 1$. If the solution is all integer then this is the optimal solution. If not, the solution is $x_1 = 1$, $x_2 = 1,\ldots,x_{r-1} = 1$ and x_r is fractional. Calculate $x_0 = \sum_{i < r} v_i$ and form $S(x_0,1,1,\ldots,1,0,0,0)$ as a lower bound to the optimal solution, where the zero components in S represent $x_i = 0$, $i \geq r$. Set R(1) = r and the r^{th} component of X to zero.

3(a). Solve (9) with the L assigned components of X added as constraints. We obtain (10). If $[Z(L + 1)] \leq x_0$

go to step 4. If $Z(L+1) > x_0$ and we have an integer solution, take $x_0 = Z(L+1)$, form a new $S(x_0,x_1,\ldots,x_n)$ from (10) and go to step 4. If $[Z(L+1)] > x_0$ and we do not have an integer solution, go to 3(b).

(b). If $x(L) > x_0$, take $x_0 = x(L)$ and form a new $S(x_0,x_1,\ldots,x_n)$ from (10) with $x_r = 0$. In any case, set $L = L + 1$, take $R(L) = r$, and set the R(L) component of X to zero and go to 3(a).

4(a). If the R(L) component of X is equal to zero change the component to one and go to 3(a). If the R(L) component of X is one go to 4(b).

(b). If $L = 1$, the optimal solution is $S(x_0,x_1,\ldots,x_n)$. If $L \neq 1$, change the R(L) component of X to two, set $L = L-1$, and go to 4(a).

This completes the algorithm. R(L) represents a routing of assignments along the branch. Only one branch is studied at a time with preference given to assigning the value of zero to the fractional variable. This allows lower bounds to be achieved more rapidly. The only permanent storage information required is the current lower bound solution S, the assigned variables R(L) at level L of the branch, and the assignment vector X.

An extension of the previous algorithm is that of Greenberg [15] in which he presents an algorithm for the solution of the integer programming problem

(11) Minimize $Z = CX$

Subject to $AX = B$

$0 \leq x_j \leq b_j$, x_j integer, $j=1,\ldots,n$,

where C is an n-vector, B is an m-vector, A is an mxn matrix and the b_j are positive integers. Any or all of the b_j may be infinite. It is assumed that the elements of C, B and A are integers.

The two phase simplex method is used to solve (11) together with the bounded variable technique of Dantzig. After solving (11) as a linear program, the condition $x_k \leq [x_k^0]$ or $x_k \geq [x_k^0] + 1$ for a basic variable x_k with its value x_k^0 as fractional is imposed. The solution for the linear program is then infeasible. Then x_k is placed at either the new upper bound $[x_k^0]$ or at a lower bound $[x_k^0] + 1$, an artificial variable equal to the amount of the infeasibility produced ($x_k^0 - [x_k^0]$ in the first case or $[x_k^0] + 1 - x_k^0$ in the second case) is added, and the new bounded variable problem is solved by the two phase procedure. If a solution is obtained to the new problem that is fractional, an additional condition $x_r \leq [x_r^1]$ or $x_r \geq [x_r^1] + 1$ is imposed for a basic variable x_r with its value x_r^1 as fractional. The new problem is then solved maintaining the constraint of x_k. This process is continued until it is necessary to backtrack which is done by simply removing the bounds on the required variables and solving the linear program to return to a previous problem. Note that bounds on the same variable may appear again; the initial bounds will automatically be satisfied.

This entire procedure enumerates possibilities for the solution to (11) in a directed tree. The rooted node corresponds to Z^0. Only one branch is investigated to the next node corresponding to the solution of (11) with say

$x_k \geq [x_k^0] + 1$ representing the branch. From each node the same procedure is repeated investigating one branch until we backtrack to a previous node that has had only one branching. For example, we would branch to a node corresponding to the solution of (11) with $x_k \leq [x_0^0]$ representing the branch when we backtrack to the rooted node. Any further backtracking to the rooted node would end the problem. This procedure is contained in the following algorithm:

Define $S(x_0,x_1,\ldots,x_n)$ as the current upper bound solution, which represents a feasible integer solution to (11) with objective value x_0.

Define I(L) as the index of the $L^{\underline{th}}$ bounded variable.

Define N(L) to show the number of branchings for the $L^{\underline{th}}$ bounded variable (Initially N(L) = 0).

Define B(L) to show the bound on the $L^{\underline{th}}$ bounded variable.

Define G(L) = 0 to indicate the $L^{\underline{th}}$ bounded variable is $\leq$ B(L), and G(L) = 1 to indicate the $L^{\underline{th}}$ bounded variable is $\geq$ B(L).

1. Set L = 0 and take $S(x_0,x_1,\ldots,x_n)$ as a feasible integer solution to (11). If none is apparent take x_0 as infinite. Solve (11) as a linear program. If the solution is all integer the problem is solved. Otherwise go to 2 maintaining the canonical form of the solution to (11).

2. Set L = L + 1. Go to 2(a).

   (a) Select a basic variable that is fractional. Suppose the variable selected is x_k with value x_k^0, set I(L) = k, N(L) = 1 and go to 2(b) or 2(c).

   (b) Set B(L) = $[x_k^0]$ and G(L) = 0. Go to 3

   (c) Set B(L) = $[x_k^0]$ + 1 and G(L) = 1. Go to 3.

3. Solve the linear program using the maintained canonical form with the new bounds on the variable x_k where k = I(L). One of the cases holds:

   (a) If the solution produces an objective value Z with [Z] + 1 $\geq$ x_0 to to 4.

   (b) If the solution produces an objective value Z with [Z] + 1 < x_0 and the solution is all integer, redefine $S(x_0,x_1,\ldots,x_n)$ as a new feasible integer solution where x_0 = Z, $x_1,\ldots,x_n$ is the new solution. Go to 4.

   (c) If the solution produces an objective value Z with [Z] + 1 < x_0 and the solution is fractional go to 2.

   (d) If the problem has an infeasible solution go to 4.

4. Set LL = L and go to 4(a).

   (a) If N(L) = 2, go to 4(b). Otherwise N(L) = 1; go to 4(c).

   (b) If L = 1, the feasible solution $S(x_0,x_1,\ldots,x_n)$ is optimal. Stop. Otherwise set N(L) = 0, L = L-1, and go to 4(a).

(c) Solve the linear programming problem starting from the current canonical form with the bounds B(L), B(L+1),...,B(LL) removed. If k = I(L) and x_k does not become a basic variable then the solution is non-unique; x_k is then made a basic variable. Go to 4(d).

(d) If G(L) = 0, set B(L) = B(L) + 1, N(L) = 2, and go to 3. Otherwise G(L) = 1; set B(L) = B(L) - 1, N(L) = 2, and go to 3.

It should be noted in this algorithm the bounds are changed systematically until the optimal integer solution is produced. The method results in a branching process that requires minimum excess computer storage over that required to solve the linear programming problem.

In addition, dynamic programming enumeration, similar to that given in [17] and developed in the dynamic programming section of this thesis, may be instituted after a fractional solution is obtained at any node to speed the calculations.

To illustrate the basic method, an example from [3] is presented where the variables are arbitrarily picked for branching and the branching is always taken with a greater than or equal sign first. The problem is to

$$\begin{aligned} \min \quad & 4x_1 + 5x_2 \\ \text{when} \quad & 3x_1 + x_2 - x_3 = 2 \\ & x_1 + 4x_2 - x_4 = 5 \\ & 3x_1 + 2x_2 - x_5 = 7 \\ & x_1, x_2, x_3, x_4, x_5 \geq 0 \text{ and integer.} \end{aligned}$$

The continuous solution produces the canonical form

$$x_1 + \frac{2}{10}x_4 - \frac{4}{10}x_5 = \frac{18}{10}$$

$$x_2 - \frac{3}{10}x_4 + \frac{1}{10}x_5 = \frac{8}{10}$$

$$x_3 + \frac{3}{10}x_4 - \frac{11}{10}x_5 = \frac{42}{10}$$

$$-Z + \frac{7}{10}x_4 + \frac{11}{10}x_5 = -\frac{112}{10} \quad .$$

Since the solution is fractional we arbitrarily select the bound $x_1 \geq 2$ added to the previous canonical form. We then obtain the canonical form

$$x_2 + \frac{1}{4}x_1 - \frac{1}{4}x_4 = \frac{5}{4}$$

$$x_3 - \frac{11}{4}x_1 - \frac{1}{4}x_4 = -\frac{3}{4}$$

$$x_5 - \frac{5}{2}x_1 - \frac{1}{2}x_4 = -\frac{9}{2}$$

$$-Z + \frac{11}{4}x_1 + \frac{5}{4}x_4 = -\frac{25}{4}$$

where $(z,x_1,x_2,x_3,x_4,x_5) = (47/4,\ 2,\ 3/4,\ 19/4,\ 0,\ 1/2)$.

We arbitrarily select $x_2 \geq 1$ and solve the previous canonical form with $x_1 \geq 2$ and $x_2 \geq 1$ to obtain the canonical form

$$x_3 - 3x_1 - x_2 = -2$$

$$x_4 - x_1 - 4x_2 = -5$$

$$x_5 - 3x_1 - 2x_2 = -7$$

$$-Z + 4x_1 + 5x_2 = 0$$

where $(z,x_1,x_2,x_3,x_4,x_5) = (13,\ 2,\ 1,\ 5,\ 1,\ 1)$, which is all integer and becomes the solution vector $S(x_0,x_1,\ldots,x_n)$.

We then remove the bound $x_2 \geq 1$ and apply the simplex procedure on the previous canonical form (e.g., Z may be decreased by decreasing x_2) and obtain the same canonical form that we obtained by selecting the bound $x_1 \geq 2$. We them impose the bound $x_2 \leq 0$ on this canonical form in addition to $x_1 \geq 2$ and obtain the canonical form

$$x_1 + 4x_2 - x_4 = 5$$

$$x_3 + 11x_2 - 3x_4 = 13$$

$$x_5 + 10x_2 - 3x_4 = 8$$

$$-Z - 11x_2 + 4x_4 = -20$$

where the solution is (20, 5, 0, 13, 0, 8). Since $Z > 13$ we remove the bounds $x_2 \leq 0$ and $x_1 \geq 2$ and solve the previous canonical form to obtain our first canonical form again. We then take $x_1 \leq 1$ and obtain the canonical form

$$x_2 + \frac{3}{2}x_1 - \frac{1}{2}x_5 = \frac{7}{2}$$

$$x_3 - \frac{3}{2}x_1 - \frac{1}{2}x_5 = \frac{7}{2}$$

$$x_4 + 5x_1 - 2x_5 = 9$$

$$-Z - \frac{7}{2}x_1 + \frac{5}{2}x_5 = \frac{-37}{2}$$

where $(z,x_1,x_2,x_3,x_4,x_5) = (14, 1, 2, 3, 4, 0)$. Sinze $Z > 13$ the problem is solved with solution $Z = 13$, $x_1 = 2$, $x_2 = 1$, $x_3 = 5$, $x_4 = 1$, $x_5 = 1$.

The advantages of this basic branching algorithm are apparent over the Land and Doig procedure even in this small example. This same problem solved using the Land and Doig

procedure as illustrated in [3] required the generation of five nodes. This algorithm required only four. Further, no additional problems are generated except that the canonical form from the original continuous solution is varied depending on the bounds on the variables. Each problem leads into the next in a natural fashion.

5. Partial Enumeration Techniques

A fourth approach, partial enumeration, is basically a branch and exclude method with various rules envoked for the purpose of exploring only certain branches of the tree.

A method for partial enumeration by Balas [1] is presented for the case where $x_j, (j=1,\ldots,n)$ equals 0 or 1. The problem to be solved should be formulated

$$(12) \quad \text{Maximize } y_0 = \sum_{j=1}^{n} c_j x_j + b_0$$

$$\text{Subject to } y_i = b_i + \sum_{j=1}^{n} a_{ij} x_j \geq 0$$

$$(i=1,\ldots,m) \text{ and } x_j = 0 \text{ or } 1, \ (j=1,\ldots,n).$$

In (12), all c_j must be non-positive. If this is not the case, x_j may be replaced by $(1-x_j)$ for any $c_j > 0$. Also, some b_i must be negative or the problem has the trivial solution with all $x_j = 0$.

There are 2^n different possible solutions. The algorithm seeks to find an optimal feasible solution or knowledge that none exists without having to evaluate each of the 2^n possible solutions. The algorithm is in the form of a tree search. A path along the tree of solutions is traced until either a new feasible solution is obtained or a node is reached which yields information that all solutions in which that particular node is included may be ruled out of consideration. The process then backtracks to the unique node that immediately precedes the one ruled out and departs that node on a different path unless none are left and it becomes necessary to backtrack further. When the process is pushed

back to the starting node and information is obtained that no other branches of the tree need be explored, the algorithm terminates.

Define S_0 to be the set of variables presently set equal to zero and S_1 as the set of variables presently assigned a value of zero or one.

Record the value of y_0 for the current best known solution. Call this value y_{0b}. Record the current values of $y_0, y_1, \ldots, y_m$ as $y_{0c}, y_{1c}, \ldots, y_{mc}$ and the maximum possible values of $y_1, y_2, \ldots, y_m$ as $y_{1m}, y_{2m}, \ldots, y_{mm}$ obtained by changing the variables in S_0 so that

$$y_{im} = y_{ic} + \sum_j \max(0, a_{ij}) \quad (i=1,\ldots,m)$$

and j is the set $\{j: \ x_j \in S_0\}$.

Initially y_{0b} may be set equal to $-\infty$ and the current subproblem has all variables in S_0. The general nature of the algorithm is:

1. Consider a subproblem. If $y_{0c} \leq y_{0b}$ abandon the subproblem. Otherwise if all $y_{ic} \geq 0$ for $(i=1,\ldots,m)$ a new best feasible solution is found. Abandon the subproblem since all $c_j \leq 0$ and it is unprofitable to assign any variable in S_0 a non-zero value.
2. Otherwise if $c_j \leq y_{0b} - y_{0c}$ for any j, put x_j in S_1 and assign it a value of zero. Revise the values of y_{im}. If and revised $y_{im} < 0$, abandon the current subproblem because it cannot have a feasible solution.

3. If $y_{im} + a_{ij} < 0$ for some i, put x_j in S_1 with an assigned value of zero and revise the y_{im}. This should be repeated for each y_{im} since a variable was last assigned to S_1 or until some $y_{im} < 0$.

4. If $y_{im} - a_{ij} < 0$ for some i, put x_j in S_1 with an assigned value of one. Revise all y_{ic} and y_{im}.

5. If no x_j is assigned a value of one in step 4, a branch must be made. A variable in S_0 is assigned to S_1 with a value of one and at the same time a new subproblem is set up that is the same as the current one except that the variable is assigned a value of zero. The choice of variable is made by taking that variable that when set equal to one increases at least one negative y_{ic} and makes the greatest decrease in the sum $\sum_{i=1}^{m} \max(0,-y_{ic})$. If two or more variables are tie, the one with the largest c_j is used.

An algorithm by Glover [10] which in its simplest form is similar to the "All Integer Method" of Gomory, but represents a weakening of the constraints of this algorithm was also a proposal for sharpening the tests of the algorithm of Balas [1] and hence reducing the size of the solution tree.

Geoffrion [8] outlined a version of the algorithm by Balas [1] which lends itself to computer implementation.

Balas' [2] filter method is an accelerated version of his earlier "additive algorithm". In the filter method a two phase approach is used where in phase I an auxiliary problem is constructed that, in phase II, is used to "filter" the solutions to which the tests of the additive algorithm are to be applied.

Clever enumerative schemes are in certain situations the best methods of solution. An algorithm for the computation of knapsack functions by Greenberg [18] falls in this category where the problem is

$$\text{(13)} \quad \text{Maximize} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{Subject to} \quad \sum_{j=1}^{n} a_j x_j = L$$

$$x_j \geq 0, \text{ and } x_j \text{ integer},$$

where each c_j is a positive number, each a_j is a positive integer, and L is a positive integer. In the usual knapsack problem the constraint in (13) is an inequality constraint. It can be assumed in this formulation that slack variables have been added to convert the inequality to equality.

The one-dimensional knapsack function is defined from (13) as:

$$\text{(14)} \quad F(x) = \max \left\{ \sum_{j=1}^{n} c_j x_j \;\middle|\; \sum_{j=1}^{n} a_j x_j = x, x_j \geq 0, \; x_j \text{ integer} \right\}.$$

If $F(x)$ can be found for all values of x and in turn the x_j found that produces the solutions, then (13) is solved when $F(L)$ is found.

The knapsack function can be written as

$$(15) \quad F(x) = \max_{j:a_j \leq x} [c_j + F(x-a_j)]$$

with initial condition $F(0) = 0$, for values of x that are feasible.

One immediate solution to (15) can be found by finding $a_r = \min a_j$, for all j. This produces $F(a_r) = c_r$. Then replace x by $x - a_r$ in (15) and obtain

$$(16) \quad F(x-a_r) = \max_{j:a_r + a_j \leq x} [c_j + F(x-a_r-a_j)] ,$$

which can then be substituted for the $F(x-a_r)$ term on the right side of (15). Another immediate solution can then be produced. Continue in this manner until $F(L)$ is found or until all values of $F(x)$ that are desired are found. Keeping track of which j produces the solution will enable us to determine the optimal x_j at the end. Only distinct values of a_j are considered. If $a_j = a_k$ and $c_j \geq c_k$ for $j \neq k$, then take $x_k = 0$.

The procedure may be summarized in the following four steps:

1. List the values of the problem as follows:

1	2	3	...	n
c_1	c_2	c_3	...	c_n
a_1	a_2	a_3	...	a_n

$x_j = 0$

Set $m = 0$. Go to 2.

2. Given the list find $a_r = \min_{a_j > m} a_j$ for columns in all sections. If $a_r = L$ go to 4. Otherwise set $m = a_r$ and go to 3.

3. Add a new section of columns to the list, if possible, as follows:

   Calculate $a'_t = a_r + a_t$ and $c'_t = c_r + c_t$ for $t = 1, \ldots, n$. Add a column headed by t if:

   (a) $a'_t \leq L$.

   (b) a'_t is not on the list.

   (c) a'_t is on the list and has a corresponding c_t value that is smaller than c'_t.

   (d) Underneath the section added write the x_j values from the section where $m = a_r$ was found. Increase x_r by one for the new section. Sections after the first may be deleted after they are no longer needed. Go to 2.

4. The problem is solved with solution c_r. The values of the variables are found below the section where $a_r = L$ appears, increase x_r by one.

This completes the algorithm. This algorithm has the computational advantage of being a single pass algorithm and does not require backtracking.

To illustrate this method the following problem is considered:

$$\text{Max } 2x_1 + \frac{10}{3}x_2 + 5x_3$$

$$\text{when } 2x_1 + 3x_2 + 4x_3 = 6.$$

$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0, \text{ all integer.}$$

1. We list the values of the problem as

1	2	3
2	$\frac{10}{3}$	5

2	3	4

$x_j = 0$

Set m = 0. Go to 2.

2. Given the above list we find $a_r = \min a_j = 2$. Set m = 2 and go to 3.

3. Write the new section as

2	3
$\frac{16}{3}$	7

5	6

$x_1 = 1.$

4. We obtain $\min a_j = 3$ in 1. No new section is added.

5. We obtain $\min a_j = 4$ in 1. No new section is added.

6. The solution is now evident in 3, where $a_j = 6$. We thus have $F(6) = 7$, $x_1 = 1$, $x_3 = 1$ as optimal.

6. Dynamic Programming Methods

The solution of a linear program is generally easier than that of an integer program. Given a linear program to be solved in integers, it is tempting to solve the linear program and then, by some process, "round" the noninteger valued variables into an optimal integer solution. Particularly appealing results of Gomory [13] have led to a dynamic programming process for rounding a linear programming solution into an optimal integer solution.

Two such approaches will be presented in this section; one for the knapsack problem and the other, a continuation of the same method, for the solution to a more general problem.

The first algorithm by Greenberg [16] is one for solving

(17) Minimize CX

Subject to $AX = B$

$$x_j \geq 0 \text{ and integer } (j=1,\ldots,n),$$

where $X = (x_1,\ldots,x_n)$, C is an n-vector, B is an m-vector, and A is an mxn matrix. It is assumed the elements of C, B, and A are integers.

The linear programming solution to (17) transforms (17) to

(18) Minimize $d + \sum_{j\in\bar{G}} c_j x_j$

Subject to $x_G + \sum_{j\in\bar{G}} \alpha_j x_j = \alpha_0$

$$x_j \geq 0 \text{ and integer } (j=1,\ldots,n),$$

where $X_G = \{x_i | i\varepsilon G\}$, $c_j \geq 0$, G is the set of indices of the basic variables and $\bar{G}$ is the set of indices of the non-basic variables. The vectors α_j (j=0 and $j\varepsilon\bar{G}$) are column vectors. The components of α_0 are non-negative. If α_0 is all integer then $x_j = 0$ ($j\varepsilon\bar{G}$), $X_G = \alpha_0$ is an optimal integer solution. If any of the α_0 are fractional then (18) may be reduced to an equivalent knapsack problem

$$(19) \quad \text{Minimize} \quad \sum_{j\varepsilon\bar{G}} c_j x_j$$

$$\text{Subject to} \quad \sum_{j\varepsilon\bar{G}} a_j x_j \equiv b \pmod{D}$$

$$x_j \geq 0 \text{ and integer,}$$

where the constraint in (19) is a congruence and D is the determinant of the coefficients of the x_i, ($i\varepsilon G$) from (17) or a reduced value. It is assumed that c_j, a_j and b are integers. The c_j in (19) are the c_j in (18) multiplied by D. The congruence in (19) is developed by finding the equation in (18), including the objective function, that has any constant, written in fractional form, having the property that its numerator and denominator are relatively prime. In this case the denominator is D. Gomory [14] shows that if we find a constant with this property then the congruence taken from the equation where the constant appears generates the congruences developed from the other equations. The cutting plane developed from the congruence represents the best cutting plane since any integer solution to the congruence produces integer solutions for the x_i ($i\varepsilon G$) in (18).

Although it may appear that the search for a constant with the property that numerator and denominator are relatively prime is a tedious one, there is a simple and rapid way to find the equation containing the constant. The revised simplex method is used to obtain the linear programming solution to (17). Therefore the inverse matrix is available to develop the constants in (18). In solving (17) the value of D is carried along as the product of the pivot elements. The desired equation is found by finding the greatest common divisor of the non-zero numerators of each of the rows of the inverse matrix. The procedure is stopped when the greatest common divisor is unity. If none of the rows produce a greatest common divisor of unity, the row with the minimum greatest common divisor is selected. The greatest common divisor of the greatest common divisors of each row also divide D thereby producing a smaller value of D by dividing. An enlarged inverse matrix is also used in considering the objective function. The procedure in [5] is used to find the greatest common divisor of the elements of each row of the inverse matrix.

Before the above procedure, the Euclidean algorithm is used to examine the α_0 and d. When a fractional constant having unity as the greatest common divisor of the numerator and denominator is found, the procedure stops. If the required row is not found then the inverse matrix is used as described above.

A problem which is given in [3], to illustrate Gomory's algorithm, is solved below by the method just described.

$$(20)\quad \min + 4x_1 + 5x_2$$

subject to
$$3x_1 + x_2 - x_3 = 2$$
$$x_1 + 4x_2 - x_4 = 5$$
$$3x_1 + 2x_2 - x_5 = 7$$
$$x_j \geq 0 \text{ and integer.}$$

The continuous solution gives us

$$(21)\quad \min + \frac{112}{10} + \frac{7}{10}x_4 + \frac{11}{10}x_5$$
$$x_1 + \frac{2}{10}x_4 - \frac{4}{10}x_5 = \frac{18}{10}$$
$$x_2 - \frac{3}{10}x_4 + \frac{1}{10}x_5 = \frac{8}{10}$$
$$x_3 + \frac{3}{10}x_4 - \frac{11}{10}x_5 = \frac{42}{10}$$

where D = 10.

Considering $d = \frac{112}{10}$, $\alpha_0 = (\frac{18}{10}, \frac{8}{10}, \frac{42}{10})$, we see that none have numerator and denominator relatively prime. The inverse matrix is

$$\begin{matrix} 0 & \frac{-2}{10} & \frac{4}{10} & 0 \\ 0 & \frac{3}{10} & \frac{-1}{10} & 0 \\ -1 & \frac{-3}{10} & \frac{11}{10} & 0 \\ 0 & \frac{-7}{10} & \frac{-11}{10} & 1 \end{matrix}$$

where the bottom row produces the coefficients in the objective function. In the first row we consider the greatest common divisor of (-2,4) which is 2. The first row produces the first constraint equation in (21). In the

second row we consider the greatest common divisor of (3,-1) which is unity. Thus the required congruence may be obtained from the second constraint in (21). Similarly, the required congruence may be obtained from the third constraint and the objective function. Using the second constraint we obtain the congruence

$$7x_4 + x_5 \equiv 8 \pmod{10} .$$

The equivalent knapsack problem becomes

$$\begin{aligned} &\text{minimize } 7x_4 + 11x_5 \\ &\text{when} \quad 7x_4 + x_5 \equiv 8 \pmod{10} \\ &\qquad x_4, x_5 \geq 0 \text{ and integer.} \end{aligned}$$

In general, once the required equation in (18) is found, e.g.,

$$x_i + \sum_{j \in \bar{G}} \alpha_{ij} x_j = \alpha_{i0} \text{ for a single } i \in G$$

(or for the objective function) the congruence is found as in Gomory [13] as

$$\sum_{j \in \bar{G}} \bar{\alpha}_{ij} x_j \equiv \bar{\alpha}_{i0} \pmod{1}$$

where $\bar{\alpha}_{ij}$ is the fractional part of α_{ij}. Multiplying through by D, we obtain

$$\sum_{j \in \bar{G}} a_j x_j \equiv b \pmod{D}$$

where $$a_j = \bar{\alpha}_{ij} D, \; b = \bar{\alpha}_{i0} D .$$

We are now interested in a solution to the equivalent knapsack problem which is in the form of (19). We can assume that no value of a_j appears more than once (if not, we can select the one with smallest c_j value). Writing

$$F(y) = \min\left[\sum_{j=1}^{n} c_j x_j \,\middle|\, \sum_{j=1}^{n} a_j x_j \equiv y \pmod{D}\right] .$$

We obtain the recursion as in [9]

$$(22) \quad F(y) = \min\,(c_j + F(y-a_j)) \quad ,$$

with $F(0) = 0$. The arguments of F are taken modulo D. We can write (22) as

$$(23) \quad F(y) = c_r + \min_j\,(c_j - c_r + F(y-a_j))$$

where $c_r = \min_j c_j$. Equation (23) has an immediate solution for $y = a_r$ and we obtain $F(a_r) = c_r$. We then write (23) as

$$(24) \quad F(y-a_r) = c_r + \min_j\,(c_j - c_r + F(y-a_r-a_j)) \; .$$

We substitute (24) for the $F(y-a_r)$ term on the right side of (23) and produce another solution to some $F(a_j)$. We stop when we have achieved a solution for F(b). We also keep track of the variable j which produces each solution and then backtrack to find the x_j. This entire procedure can be summarized in the following algorithm:

1. List the values of the problem as follows:

1	2	3	...	n
c_1	c_2	c_3	...	c_n
a_1	a_2	a_3	...	a_n

   Go to 2.

2. Given the list

1	2	3	...	n	t_1	t_2	...	t_s
c_1	c_2	c_3	...	c_n	c'_{t_1}	c'_{t_2}	...	c'_{t_s}
a_1	a_2	a_3	...	a_n	a'_{t_1}	a'_{t_2}	...	a'_{t_s}

find $c_r = \min c_j$ for all unmarked columns. Set $F(a_r) = c_r$ and $I(a_r) = r$, where a_r and r are corresponding elements to c_r in the column. If $a_r = b$ go to 4. Otherwise mark the column if it is one of the first n. Go to 3.

3. Add columns to the list, if possible, as follows: Calculate $a'_t \equiv a_r + a_t \pmod{D}$ and $c'_t = c_r + c_t$ for $t = 1,\ldots,n$. Add a column headed by t if:
   (a) $F(a'_t)$ has not been found.
   (b) a'_t is not on the list.
   (c) a'_t is on the list and has a corresponding c_t value that is larger than c'_t.

   Delete columns (beyond the first n) having elements a'_t
   (d) after $F(a'_t)$ is found.
   (e) after a new a'_t is found as in 3(c).

   Go to 2.

4. The problem is solved with solution F(b). The values of the variables are found as follows:
   (a) Set $x_j = 0$, $j = 1,\ldots,n$ and $y = b$. Go to 4(b).
   (b) Look up $I(y) = j$. Set $x_j = x_j + 1$. Go to 4(c).
   (c) Set $y \equiv y - a_j \pmod{D}$. If $y = 0$ go to 5. Otherwise go to 4(b).

5. End. The final x_j values are the required ones.

This completes the algorithm. If solutions are desired for all values of y then the algorithm can be continued instead of stopping when b is reached. Continuing with our example and using the algorithm of Greenberg [16], the

equivalent knapsack problem is

$$\text{Minimize} \quad 7x_4 + 11x_5$$

$$\text{Subject to} \quad 7x_4 + x_5 \equiv 8 \pmod{10}$$

$$x_4, x_5 \geq 0 \text{ and integer,}$$

we list

4	5	
7	11	
7	1	.

Thus 7 = min c_j, F(7) = 7, I(7) = 4. The list becomes (Marking the first column)

4	5	4	5
7	11	14	18
7 *	1	4	8

Thus F(1) = 11, I(1) = 5. The list becomes

4	5	4	5	5
7	11	14	18	22
7 *	1 *	4	8	2

F(4) = 14, I(4) = 4. The list becomes

4	5	4	5	5
7	11	18	22	25
7 *	1 *	8	2	5

F(8) = 18, I(8) = 4. We stop.

Backtracking I(8) = 4, $x_4 = 1$

I(1) = 5, $x_5 = 1$, end.

The solution is F(8) = 18, $x_4 = 1$, $x_5 = 1$.

Approximately 25 problems have been tried using this method [16] where there was a maximum of 2000 variables and 150 equations. The method was used to solve integer programs arising as covering problems. Usually the method produces an immediate integer solution. If the knapsack solution does not produce feasible (positive) integers to the problem, success has always been obtained by adding a single cut.

The above algorithm has been extended by Greenberg [17] for integer solutions to bounded variable linear programs. As previously, the continuous linear programming solution is first found. If the continuous solution is fractional then a linear congruence is added as a constraint. The problem is then put into a dynamic programming format and the optimal integer solution found.

The algorithm presented is for solving

(25) Minimize CX

Subject to $AX = B$

$0 \leq x_j \leq b_j$, x_j integer, $j=1,\ldots,n$,

where $X = (x_1,\ldots,x_n)$, C is an n-vector, B is an m-vector, A is an mxn matrix and the b_j are integers. It is assumed the elements of C, B, and A are integers. In this formulation the b_j may be infinite. This algorithm also includes the case where the x_j are restricted to be either zero or one.

A bounded variable linear programming technique is used to obtain the linear programming solution to (25). This transforms (25) to

(26) Minimize $d + \sum_{j\varepsilon\bar{G}} c_j x_j$

$$X_G + \sum_{j\varepsilon\bar{G}} \alpha_j x_j = \alpha_0,$$

$$0 \leq x_j \leq b_j,\ x_j \text{ integer, } j=1,\ldots,n,$$

where the vector $X_G = \{x_i \ i\varepsilon G\}$, G is the set of indices of the basic variables, and $\bar{G}$ is the set of indices of the non-basic variables. Further, some of the non-basic variables may be at their upper bounds in the continuous solution of (25). For those variables their corresponding c_j values are non-positive. Also, $c_j \geq 0$ for non-basic variables that are at the zero value. For ease of computation, we make the transformation $x_j' = b_j - x_j$ for those non-basic variables at their upper bound. Thus we may assume a form like (26) with all $c_j \geq 0$ and all non-basic variables at zero values. The vectors α_j ($j=0$ and $j\varepsilon\bar{G}$) are column vectors. The components of α_0 are non-negative. If α_0 is all integer then $x_j = 0$ ($j\varepsilon\bar{G}$), $X_G = \alpha_0$ is an optimal solution (if any of the x_j, $j\varepsilon\bar{G}$ are really the x_j' under the transformation, then $x_j = b_j$). If any of the α_0 are fractional, problem (26) is converted to the problem:

(27) Minimize $\sum_{j\varepsilon\bar{G}} c_j x_j$

when $\sum_{j\varepsilon\bar{G}} \alpha_j x_j \leq \alpha_0$

$$\sum_{j\varepsilon\bar{G}} a_j x_j \equiv b \pmod{D}$$

$$0 \leq x_j \leq b_j,\ x_j \text{ integer, } j\varepsilon\bar{G}.$$

The inequality constraints in (27) are obtained from (26) by dropping the x_i, $i \in G$. D is the determinant of the coefficients of the x_i, $i \in G$ from (25). The value of D may be found as the product of the pivot elements during the simplex procedure for obtaining the continuous solution of (25). The linear congruence in (27) is a single congruence where the a_j and b are integers. The congruence results from the requirement that all x_i, $i \in G$, be integers.

After finding the congruence in (27) a dynamic programming formulation of (27) is developed. Writing

$$F(x,\alpha,\beta) = \min\left[\sum_{j\in\bar{G}} c_j x_j \,\middle|\, \sum_{j\in\bar{G}} a_j x_j \leq \alpha\ ,\ \sum_{j\in\bar{G}} a_j x_j \equiv x \pmod{D},\ 0 \leq e_j x_j \leq \beta\right]$$

the dynamic programming recursion

$$(28)\quad F(x,\alpha,\beta) = \min_{j\in\bar{G}} \left(c_j + F(x-a_j,\alpha-\alpha_j,\beta-e_j)\right)$$

$$F(0,\alpha,\beta) = 0 \text{ for } \alpha \geq 0,\ \beta \geq 0,$$

is obtained. The x argument of F is taken modulo D. The vector e_j is composed of a unit element with the other elements zero. The unit element corresponds to the unit coefficient of x_j in $x_j \leq b_j$. It is assumed that only feasible values of x, α, and β are used in (28).

One immediate solution to (28) can be obtained by finding $c_r = \min c_j$. This produces $F(a_r,\alpha_r,e_r) = c_r$. Then replace x by $x - a_r$, α by $\alpha - \alpha_r$, and β by $\beta - e_r$ in (28) and substitute the result for the $F(x-a_r,\alpha-\alpha_r,\beta-e_r)$ term on the right side of (28), then another immediate solution can be produced. Continue this way until $F(b,\alpha,\beta)$ is found

where $0 \leq \alpha_0 - \alpha \leq b_G$ and $\beta \leq b_{\bar{G}}$ where the vector $b_G = \{b_i | i \varepsilon G\}$ and the vector $b_{\bar{G}} = \{b_i | i \varepsilon \bar{G}\}$. This entire procedure is contained in the following algorithm:

1. Suppose the indices in $\bar{G}$ are $(1,2,\ldots,m)$, list the values of the problem as

1	2	3	...	m
c_1	c_2	c_3	...	c_m
α_1	α_2	α_3	...	α_m
a_1	a_2	a_3	...	a_m

$x_j = 0$

Go to 2.

2. Given the list, find $c_r = \min c_j$ for all unmarked columns in all sections. If $a_r = b$ and $0 \leq \alpha_0 - \alpha_r \leq b_G$ go to 4. Otherwise mark the column and go to 3.

3. Add a new section of columns to the list, if possible, as follows:

   (a) calculate $c_j' = c_r + c_j$, $\alpha_j' = \alpha_r + \alpha_j$, $a_j' \equiv a_r + a_j \pmod{D}$ for all values of $j \varepsilon \bar{G}$ (i.e., values j, c_j, α_j, a_j are taken from the list in step 1.) where $x_j < b_j$ for $j \neq r$ and where $x_r + 1 < b_r$ for $j = r$ for the section containing the newly marked column.

   (b) add the column headed by j in the new section with values c_j', α_j' and a_j'.

(c) Underneath the section added write the x_j values from the section containing the newly marked column. Increase x_r by one for the section. When columns in new sections become marked they may be deleted after the calculations are made and the new section is added to the list. A new column may already be on the list. It need not be added if the x_j values in corresponding sections would coincide (after increasing the x_j values by one for the j heading values of the two columns). Go to 2.

4. The integer problem is solved with objective function value $d + c_r$. The values of the basic variables in (26) are $X_G = \alpha_0 - \alpha_r$. The values of the non-basic variables are found below the section where $a_r = b$ appears; increase x_r by one.

This completes the algorithm. The algorithm may be simplified by calculating α'_j only when $a'_j = b$ by using the proper x_j values and the α_j from step 1. An integer solution is sure to be found because all integer solutions to the congruence in (27) are systematically produced in order of increasing objective function value.

To illustrate this procedure we use the problem given in [2] as

$$\text{Minimize} \quad 5x_1 + 7x_2 + 10x_3 + 3x_4 + x_5$$

$$\text{when} \quad x_1 - 3x_2 + 5x_3 + x_4 - 4x_5 \geq 2$$

$$-2x_1 + 6x_2 - 3x_3 - 2x_4 + 2x_5 \geq 0$$

$$- x_2 + 2x_3 - x_4 - x_5 \geq 1$$

$$0 \leq x_j \leq 1, \; x_j \text{ integer}, \; j=1,\ldots,5.$$

We introduce surplus variables $x_6 \geq 0$, $x_7 \geq 0$, and $x_8 \geq 0$ (i.e., with ∞ as upper bounds) and find the continuous solution and equivalent problem:

(29) Minimize

$$9 + \tfrac{93}{9}x_1 + \tfrac{156}{9}x_4 + \tfrac{42}{9}x_5 + \tfrac{24}{9}x_7 + \tfrac{137}{9}x_8$$

when

$$-\tfrac{7}{9}x_1 - \tfrac{28}{9}x_4 + \tfrac{13}{9}x_5 + x_6 + \tfrac{1}{9}x_7 - \tfrac{1}{3}x_8 = \tfrac{1}{3}$$

$$-\tfrac{2}{9}x_1 + x_3 - \tfrac{8}{9}x_4 - \tfrac{4}{9}x_5 - \tfrac{1}{9}x_7 - \tfrac{2}{3}x_8 = \tfrac{2}{3}$$

$$-\tfrac{4}{9}x_1 + x_2 - \tfrac{7}{9}x_4 + \tfrac{1}{9}x_5 - \tfrac{2}{9}x_7 + \tfrac{1}{3}x_8 = \tfrac{1}{3}$$

$$0 \leq x_j \leq 1, \; x_j \text{ integer}, \; j=1,\ldots,5$$

$$0 \leq x_j, \; j=6, 7, 8.$$

We have D = 9. The second equation produces the congruence

$$7x_1 + x_4 + 5x_5 + 8x_7 + 3x_8 \equiv 6 \pmod 9$$

which is added as a constraint. Multiplying through by 9 where applicable, we list

(30)

	1	4	5	7	8
9	93	156	42	24	137
3	-7	-28	13	1	-3
6	-2	-8	-4	-1	-6
3	-4	-7	1	-2	3
6	7	1	5 *	8 *	3

$x_1 = 0$

For convenience we include at the left of the list the objective function value, the right hand sides (multiplied by 9) for the equalities in (29), and the right hand side for the congruence. We have $24 = \min c_j$, i.e., $c_r = 24$, with $r = 7$. We add the section

(31)

1	4	5	7	8
117	180	66	48	161
-6	-27	14	2	-2
-3	-9	-5	-2	-7
-6	-9	-1	-4	1
6	0	4 *	7 *	2

$x_7 = 1$

We mark the 7 column in (30). We have $\min c_j = 42$ from ( ); (30); we add the section

(32)

1	4	8
135	198	178
6	-15	10
-6	-12	-10
-3	-6	4
3	6	8

$x_5 = 1$

We need not add a column headed by 7 in (32) because it would duplicate the 5 column in (31) (the use of either column would require $x_5 = 1$ and $x_7 = 1$). We do not add a 5 column in (32) since x_5 would be at its upper bound in (32). We mark the 5 column in (30). We have min $c_j = 48$ in (31). We add the section

(33)

1	4	5	7	8
141	204	90	72	185
-5	-26	15	3	-1
-4	-10	-6	-3	-8
-8	-11	-3	-6	-1
5	8	3	6	1

$x_7 = 2$

We mark the 7 column in (31). We have min $c_j = 66$ from (31). We add the section

(34)

1	4	8
159	222	203
-7	-14	11
-7	-13	-11
-5	-8	2
2	5	7

$x_1 = 1, x_5 = 1$

We mark the 5 column in (31). The solution is now apparent in the 7 column in (33). We have the optimal solution $x_7 = 3$, $x_6 = (3-3)/9 = 0$, $x_3 = (6+3)/9 = 1$, $x_2 = (3+6)/9 = 1$; the value of the objective function is $9 + 72/9 = 17$.

7. Summary

Since 1954, the field of integer programming has made rapid progress. Many new approaches have been tried on various type problems with varying amounts of success. The developments in computers have enabled many problems to be solved today that fifteen years ago could not be handled. Many computer codes have been developed around the algorithms presented here. Information on computational experience with these codes is scarce and publications in this area are lacking. The little information that is available indicates there is much to be tried and certainly room for clever imaginative schemes for solving this type of problem. Possibly the best approach or method for solution will be a scheme employing all of the categories described in this thesis.

BIBLIOGRAPHY

1. Balas, E., "An Additive Algorithm for Solving Linear Programs with Zero-One Variables", Operations Research, Vol. 13 (1965), pp. 517-546.

2. _____ , "Discrete Programming by the Filter Method", Operations Research. Vol. 15 (1967), pp. 915-957.

3. Balinski, M. L., "Integer Programming: Methods, Uses, Computation", Management Science, Vol. 12 (1965), pp. 253-313.

4. Benders, J. F., "Partitioning Procedures for Solving Mixed-Variables Programming Problems", Numerische Mathematik, Vol. 4 (1962), pp. 238-252.

5. Blankinship, W. A., "A New Version of The Euclidean Algorithm", American Mathematical Monthly, Vol. 70 (1963), pp. 742-745.

6. Dantzig, G. B., "Discrete Variable Extremum Problems", Operations Research, Vol. 5 (1957), pp. 266-277.

7. _____ , Fulkerson, D. R. and Johnson, S. M., "Solutions of a Large Scale Traveling Salesman Problem", Journal of the Operations Research Society of America, Vol. 2 (1954), pp. 393-410.

8. Geoffrion, A. M., "Integer Programming by Implicit Enumeration and Balas' Method", SIAM Review, Vol. 9 (1967), pp. 178-190.

9. Gilmore, P. C., and Gomory, R. E., "The Theory and Computation of Knapsack Functions", Operations Research, Vol. 14 (1966), pp. 1045-1074.

10. Glover, F., "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem", Operations Research, Vol. 13 (1965), pp. 879-919.

11. Gomory, R. E., "Outline of an Algorithm for Integer Solutions to Linear Programs", Bulletin of the American Mathematical Society, Vol. 64 (1958), pp. 275-278.

12. _____ , "An Algorithm for the Mixed Integer Problem", Rand Corporation, RM-2597 (1960).

13. _____ , "On the Relation Between Integer and Non-Integer Solutions to Linear Programs", Proc. Nat. Acad. Science, Vol. 53 (1965), pp. 260-265.

14. _____, "An Algorithm for Integer Solutions to Linear Programs", pp. 269-302 in Recent Advances in Mathematical Programming, R. L. Graves and P. Wolfe (Eds.), McGraw-Hill, New York, 1963.

15. Greenberg, H., "The Solution of Integer Programs by Bounded Variable Techniques", United States Naval Postgraduate School, Report 55 Gd 8051A, 1968.

16. _____, "Knapsack Solutions In Integer Programming", United States Naval Postgraduate School, Report 55 Gd 8051A, 1968.

17. _____, "An Algorithm for Integer Solutions to Bounded Variable Linear Programs", United States Naval Postgraduate School, Report 55 Gd 8051A, 1968.

18. _____, "An Algorithm for the Computation of Knapsack Functions", United States Naval Postgraduate School, Report 55 Gd 8051A, 1968.

19. Hegerich, R. L., "A Branch and Exclude Algorithm for the Knapsack Problem", Master's Thesis, The United States Naval Postgraduate School, Monterey, California, 1968.

20. Land, A. H., and Doig, A. G., "An Automatic Method of Solving Discrete Programming Problems", Econometrica, Vol. 28, (1960), pp. 497-520.

21. Young, R. D., "A Primal (All Integer) Integer Programming Algorithm", Journal of Research, Vol. 69B (1965), pp. 213-250.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	20
2.	Library Naval Postgraduate School Monterey, California 93940	2
3.	Director, Systems Analysis Division (OP96) Office of the Chief of Naval Operations Washington, D. C. 20350	1
4.	Department of the Army Civil Schools Branch, OPO, OPD Washington, D. C. 20315	1
5.	Professor Harold Greenberg Department of Operations Analysis Naval Postgraduate School Monterey, California 93940	1
6.	Captain Charles W. Hobart 302 Ardennes Circle Fort Ord, California 93941	1
7.	Department of Operations Analysis Naval Postgraduate School Monterey, California 93940	1

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE
ON INTEGER LINEAR PROGRAMMING

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Thesis

5. AUTHOR(S) *(Last name, first name, initial)*
HOBART, Charles Wendell, Captain, USA

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1968	58	21

8a. CONTRACT OR GRANT NO.

b. PROJECT NO.

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. AVAILABILITY/LIMITATION NOTICES
[redacted]

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY
Naval Postgraduate School
Monterey, California

13. ABSTRACT

A survey of the methods of solving the integer program, $\max \sum_{j=1}^{n} c_j x_j$ subject to $\sum_{j=1}^{n} a_{ij} x_j = b_i$ $(i=1,\ldots,m)$ and $x_j \geq 0$ and integer $(j=1,\ldots,n)$ is presented. Emphasis is placed on methods developed since 1960 with many as yet unpublished methods presented. Examples are given for the unpublished methods.

DD FORM 1 JAN 64 1473

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Integer Programming						
Knapsack Functions						
Dynamic Programming						
Bounded Variables						